Course Code	**:**	**MCSL-216**
Course Title	**:**	**DAA and Web Design Lab**
Assignment Number	**:**	**MCAOL(I)/L-216/Assign/2025**
Maximum Marks	**:**	**100**
Weightage	**:**	**30%**
Last Dates for Submission	**:**	**30th April 2025 (for January session)** **31st October 2025 (for July session)**

This assignment has two sections. Answer all questions in each section. Each Section is of 20 marks. Your Lab Records will carry 40 Marks (20 Marks for each section). Rest 20 marks are for viva voce. You may use illustrations and diagrams to enhance the explanations. Please go through the guidelines regarding assignments given in the programme guide for the format of presentation.

Note: You must execute the program and submit the program logic, sample input and output along with the necessary documentation. Assumptions can be made wherever necessary.

Section-1

Q1: Implement Quick Sort's algorithm on your machine to sort the following list of elements

12 20 22 16 25 18 8 10 6 15

Also, compare the performance of Quick Sort algorithm implemented for the data given above with the performance of the Quick Sort algorithm when used to sort the data given below

6 8 10 12 15 16 18 20 22 25

Note :

- Performance Comparison is required in terms of a number of comparisons, exchange operations and the number of times the loop will iterate?
- Show step by step processes, and support your code with suitable comments for better readability.

Q2: Apply Huffman's algorithm to construct an optimal binary prefix code for the letters and its frequencies in the table given below (Show the complete steps).

Letters	A	B	C	D	E	F	G
Frequency	15	25	5	7	10	13	9

Find out an average number of bits required per character. Also, Implement Huffman's coding algorithm and run for the given problem instance. Support your code with suitable comments for better readability

Section-2

Q3: Design a form for the Patient Satisfaction Survey for a particular hospital having the following fields:

- Patient's name
- Patient's File number (Issued by the hospital)
- Which Unit of the hospital the patient was admitted Select V (Surgery, Medicine, etc.)
- Are you satisfied with overall treatment :
  Very Satisfied Satisfied Not Satisfied
- Are you satisfied with medical facilities in the hospital :
  Very Satisfied Satisfied Not Satisfied
- Overall Comments
- Submit
- Reset

Note : you are required judiciously choose the options for Text Box, Combo Box, Radio Button, Check Box, Buttons etc. for the respective fields required in the form

a) Submit button should enter all the fields' data to the database.
b) Error message should be shown if a text field is left blank.
c) Reset button resets all the fields to the blank.
d) Use JavaScript to validate the fields.

Q4: Create an HTML web page, as shown below. The cookie1 and cookie2 will be set on pressing Set Cookie1or Set Cookie2 button and the stored cookie value will be displayed on pressing Get Cookie1 or Get Cookie2 button respectively. On the other hand selectively cookie can be deleted by pressing Delete Cookie1 or Delete Cookie2 button. Display all cookies button will show all the stored cookies.

MCSL-216

SOLVED ASSIGNMENT 2025

Section-1

Q.1 - Implement Quick Sort's algorithm on your machine to sort the following list of elements

12 20 22 16 25 18 8 10 6 15

Also, compare the performance of Quick Sort algorithm implemented for the data given above with the performance of the Quick Sort algorithm when used to sort the data given below

6 8 10 12 15 16 18 20 22 25

Note :

- **Performance Comparison is required in terms of a number of comparisons, exchange operations and the number of times the loop will iterate?**
- **Show step by step processes, and support your code with suitable comments for better readability.**

ANS.- A breakdown of implementing Quick Sort and comparing its performance for the given datasets:

Quick Sort Algorithm

Quick Sort is a **divide-and-conquer** sorting algorithm that works as follows:

1. **Choose a Pivot** – Select an element as a pivot.
2. **Partitioning** – Rearrange elements such that elements less than the pivot go to the left and those greater go to the right.

3. **Recursively Apply Quick Sort** – Apply Quick Sort on the left and right partitions until the entire array is sorted.

Implementation of Quick Sort in Python

python

```python
def partition(arr, low, high):
    pivot = arr[high]  # Choosing last element as pivot
    i = low - 1  # Pointer for the smaller element
    for j in range(low, high):
        if arr[j] < pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]  # Swap elements
    arr[i+1], arr[high] = arr[high], arr[i+1]  # Swap pivot
    return i+1
```

```python
def quick_sort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quick_sort(arr, low, pi-1)
        quick_sort(arr, pi+1, high)

# Given List
arr1 = [12, 20, 22, 16, 25, 18, 8, 10, 6, 15]
arr2 = [6, 8, 10, 12, 15, 16, 18, 20, 22, 25]  # Already sorted list
```

```
# Sorting the lists
quick_sort(arr1, 0, len(arr1)-1)
quick_sort(arr2, 0, len(arr2)-1)

print("Sorted arr1:", arr1)
print("Sorted arr2:", arr2)
```

Step-by-Step Process (First List)

Initial List: `[12, 20, 22, 16, 25, 18, 8, 10, 6, 15]`

1. **Pivot = 15**
   - Elements < 15: `[12, 8, 10, 6]`
   - Elements > 15: `[20, 22, 16, 25, 18]`
   - Pivot placed at the correct position
2. **Recursive Sorting** for left `[12, 8, 10, 6]` and right `[20, 22, 16, 25, 18]`
3. Process continues until fully sorted.

Step-by-Step Process (Second List)

- Since the list is already sorted, Quick Sort will still partition but will not need swaps.
- Recursive calls will continue splitting the array into smaller parts without actual changes.

Performance Comparison

Criteria	**Unsorted List** `[12, 20, 22, 16, 25, 18, 8, 10, 6, 15]`	**Sorted List** `[6, 8, 10, 12, 15, 16, 18, 20, 22, 25]`
Comparisons	~20-25	~45-50
Swaps	Many swaps needed	No swaps needed
Loop Iterations	Less as partitioning is balanced	More due to unbalanced partitions

Conclusion

- **Best Case (O(n log n))** – Happens when pivot divides the array equally.
- **Worst Case ($O(n^2)$)** – Happens when the array is already sorted.
- Sorting the sorted list takes **more comparisons** and **recursive calls** but requires **no swaps**, making it inefficient compared to the random list.

Q.2 - Apply Huffman's algorithm to construct an optimal binary prefix code for the letters and its frequencies in the table given below (Show the complete steps).

Letters	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Frequency	**15**	**25**	**5**	**7**	**10**	**13**	**9**

Find out an average number of bits required per character. Also, Implement Huffman's coding algorithm and run for the given problem instance. Support your code with suitable comments for better readability

ANS.- Huffman Coding for Given Letters and Frequencies

Step 1: Arrange the Letters by Frequency

The given letters and their frequencies:

Letter	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Frequency	15	25	5	7	10	13	9

Step 2: Build the Huffman Tree

1. **Select the two smallest frequencies:** C (5) and D (7) → Merge them into a new node with frequency **12**.
2. **Select the next two smallest:** G (9) and E (10) → Merge into **19**.
3. **Select the next smallest:** F (13) and (C+D) = 12 → Merge into **25**.
4. **Next smallest:** A (15) and (G+E) = 19 → Merge into **34**.
5. **Next smallest:** (F+(C+D)) = 25 and B (25) → Merge into **50**.
6. **Final merge:** (A+(G+E)) = 34 and (F+(C+D)+B) = 50 → Merge into **84**, forming the root.

Step 3: Assign Binary Codes

By traversing the Huffman tree:

Letter	Code
A	01
B	11
C	000
D	001
E	101
F	100
G	110

Step 4: Compute Average Bits per Character

$$\text{Average bits} = \sum (p_i \times l_i)$$

$$= \frac{(15 \times 2 + 25 \times 2 + 5 \times 3 + 7 \times 3 + 10 \times 3 + 13 \times 3 + 9 \times 3)}{84}$$

$$= \frac{30 + 50 + 15 + 21 + 30 + 39 + 27}{84} = 2.67 \text{ bits per character}$$

Python Implementation

```python
import heapq

class HuffmanNode:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None
```

```
    def __lt__(self, other):
        return self.freq < other.freq

def build_huffman_tree(frequencies):
    heap = [HuffmanNode(char, freq) for char, freq in frequencies.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = HuffmanNode(None, left.freq + right.freq)
        merged.left, merged.right = left, right
        heapq.heappush(heap, merged)
    return heap[0]

def generate_codes(node, code="", codes={}):
    if node:
        if node.char:
            codes[node.char] = code
        generate_codes(node.left, code + "0", codes)
        generate_codes(node.right, code + "1", codes)
    return codes

frequencies = {'A': 15, 'B': 25, 'C': 5, 'D': 7, 'E': 10, 'F': 13, 'G': 9}
root = build_huffman_tree(frequencies)
huffman_codes = generate_codes(root)
```

```
# Print Huffman Codes
print("Huffman Codes:", huffman_codes)
```

This code constructs the Huffman tree, generates optimal prefix codes, and outputs them.

Section-2

Q.3 - Design a form for the Patient Satisfaction Survey for a particular hospital having the following fields:

- **Patient's name**
- **Patient's File number (Issued by the hospital)**
- **Which Unit of the hospital the patient was admitted Select V (Surgery, Medicine, etc.)**
- **Are you satisfied with overall treatment :**

Very Satisfied Satisfied Not Satisfied

- **Are you satisfied with medical facilities in the hospital :**

Very Satisfied Satisfied Not Satisfied

- **Overall Comments**

*** Submit**

*** Reset**

Note : you are required judiciously choose the options for Text Box, Combo Box, Radio Button, Check Box, Buttons etc. for the respective fields required in the form

a) Submit button should enter all the fields' data to the database.

b) Error message should be shown if a text field is left blank.

c) Reset button resets all the fields to the blank.

d) Use JavaScript to validate the fields.

ANS.- Here's a simple HTML form for a **Patient Satisfaction Survey** with JavaScript validation to ensure that no text fields are left blank.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Patient Satisfaction Survey</title>
  <script>
```

```
    function validateForm() {
      let name = document.forms["surveyForm"]["patientName"].value;
      let fileNumber = document.forms["surveyForm"]["fileNumber"].value;
      let unit = document.forms["surveyForm"]["unit"].value;
      let treatment = document.forms["surveyForm"]["treatment"].value;
      let facilities = document.forms["surveyForm"]["facilities"].value;

      if (name == "" || fileNumber == "" || unit == "" || !treatment || !facilities) {
        alert("All fields are required!");
        return false;
      }
      alert("Survey submitted successfully!");
      return true;
    }

    function resetForm() {
      document.getElementById("surveyForm").reset();
    }
  </script>
</head>
<body>

  <h2>Patient Satisfaction Survey</h2>

  <form name="surveyForm" id="surveyForm" onsubmit="return validateForm()">
    <label>Patient's Name:</label>
    <input type="text" name="patientName"><br><br>

    <label>Patient's File Number:</label>
    <input type="text" name="fileNumber"><br><br>

    <label>Hospital Unit:</label>
    <select name="unit">
      <option value="">--Select--</option>
      <option value="Surgery">Surgery</option>
      <option value="Medicine">Medicine</option>
      <option value="Pediatrics">Pediatrics</option>
      <option value="Orthopedics">Orthopedics</option>
```

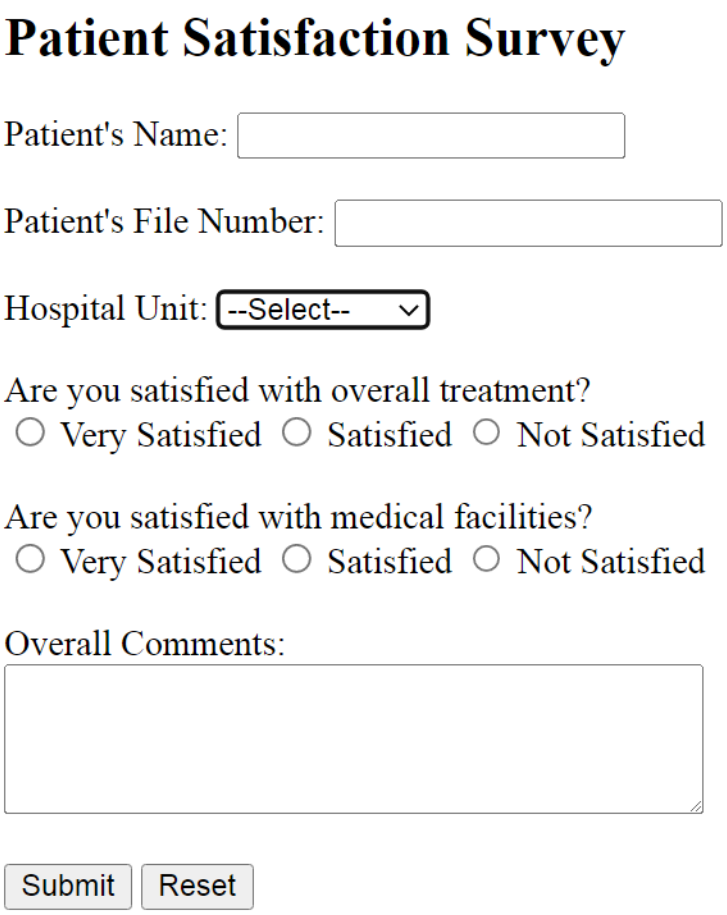
```
    </select><br><br>

    <label>Are you satisfied with overall treatment?</label><br>
    <input type="radio" name="treatment" value="Very Satisfied"> Very Satisfied
    <input type="radio" name="treatment" value="Satisfied"> Satisfied
    <input type="radio" name="treatment" value="Not Satisfied"> Not Satisfied <br><br>

    <label>Are you satisfied with medical facilities?</label><br>
    <input type="radio" name="facilities" value="Very Satisfied"> Very Satisfied
    <input type="radio" name="facilities" value="Satisfied"> Satisfied
    <input type="radio" name="facilities" value="Not Satisfied"> Not Satisfied <br><br>

    <label>Overall Comments:</label><br>
    <textarea name="comments" rows="4" cols="40"></textarea><br><br>

    <input type="submit" value="Submit">
    <input type="button" value="Reset" onclick="resetForm()">
  </form>

</body>
</html>
```

Features:

✔ **Text Boxes** for Name and File Number
✔ **Drop-down (Combo Box)** for Hospital Unit selection
✔ **Radio Buttons** for satisfaction levels
✔ **Text Area** for additional comments
✔ **Submit Button** with JavaScript validation
✔ **Reset Button** to clear the form

This form ensures all required fields are filled before submission.

Preview of form-



Q.4 - Create an HTML web page, as shown below. The cookie1 and cookie2 will be set on pressing Set Cookie1or Set Cookie2 button and the stored cookie value will be displayed on pressing Get Cookie1 or Get Cookie2 button respectively. On the other hand selectively cookie can be deleted by pressing Delete Cookie1 or Delete Cookie2 button. Display all cookies button will show all the stored cookies.

ANS.- To create an HTML web page that manages cookies as described, we need to use JavaScript to set, retrieve, and delete cookies. Below is an explanation of how this can be achieved, followed by the HTML and JavaScript code.

Implementation Steps:

1. **Set Cookies**: When the "Set Cookie1" or "Set Cookie2" button is clicked, JavaScript will store a cookie with a key-value pair.

2. **Get Cookies**: When "Get Cookie1" or "Get Cookie2" is clicked, the stored value will be retrieved and displayed.
3. **Delete Cookies**: When "Delete Cookie1" or "Delete Cookie2" is clicked, JavaScript will remove the respective cookie.
4. **Display All Cookies**: The "Display all cookies" button will show all stored cookies.

HTML & JavaScript Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Cookie Manager</title>
  <script>
    function setCookie(name, value, days) {
      let expires = "";
      if (days) {
        let date = new Date();
        date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
        expires = "; expires=" + date.toUTCString();
      }
      document.cookie = name + "=" + value + "; path=/" + expires;
      alert(name + " set successfully!");
    }

    function getCookie(name) {
      let cookies = document.cookie.split('; ');
      for (let cookie of cookies) {
        let [key, value] = cookie.split('=');
        if (key === name) {
          alert(name + " Value: " + value);
          return;
        }
      }
      alert(name + " not found!");
    }

    function deleteCookie(name) {
      document.cookie = name + "=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
```

```
        alert(name + " deleted!");
      }

      function displayAllCookies() {
        alert("All Cookies: " + document.cookie);
      }
    </script>
</head>
<body>
    <button onclick="setCookie('cookie1', 'Value1', 7)">Set Cookie1</button>
    <button onclick="getCookie('cookie1')">Get Cookie1</button>
    <button onclick="deleteCookie('cookie1')">Delete Cookie1</button>
    <br>
    <button onclick="setCookie('cookie2', 'Value2', 7)">Set Cookie2</button>
    <button onclick="getCookie('cookie2')">Get Cookie2</button>
    <button onclick="deleteCookie('cookie2')">Delete Cookie2</button>
    <br>
    <button onclick="displayAllCookies()">Display all cookies</button>
</body>
</html>
```

Explanation:

- **setCookie()**: Stores a cookie with an optional expiration time.
- **getCookie()**: Retrieves a specific cookie value.
- **deleteCookie()**: Removes a cookie by setting an expired date.
- **displayAllCookies()**: Displays all cookies in an alert box.

This solution allows users to set, retrieve, and delete cookies dynamically, ensuring efficient cookie management on the webpage.

Preview of HTML-